Provenance-Linked Diff Review as a Substrate for Diagnosing Behavioral Change

mowa

TECHNICAL DESIGN NOTE · POST-DEPLOYMENT REVIEW

ABSTRACT

When a prompt-controlled large language model application behaves differently than expected, the operator needs to determine what changed and when. This note describes the post-merge diagnostic surface mowa provides today: a per-prompt history timeline that aligns mowa-side edits, scan-detected drift, pull request open/close/merge events, and authored manual edits on a single axis, with a side-by-side Compare view between any two versions. We outline what the current surface answers, the questions it does not yet answer, and the design path from human-driven diff review toward automated counterfactual attribution.

Setting

Behavioral changes in prompt-controlled systems can originate from at least four sources: an explicit prompt revision; a silent provider-side model update; a shift in the distribution of upstream inputs; or an interaction among these. The operator's first practical question after observing a behavioral incident is not theoretical causality — it is the much narrower *what changed in our control surface, and when*.

mowa is positioned at this narrower question. The product owns the version history of the prompt artifact; it does not own the production output stream. The diagnostic surface described here is correspondingly scoped to what can be answered from version history alone, with the explicit understanding that the broader counterfactual question requires data this product does not currently collect.

History Timeline

Each tracked prompt has a History tab that aligns three event streams on a single axis: **version rows** (every distinct content state ever recorded for the prompt), **suggestion lifecycle** events (a PR opened, reviewed, closed, or merged), and **provenance transitions** (the current pointer moving, the GitHub pointer moving, an external commit landing that the system detected via the push webhook).

An entry shows version number, source label, author, timestamp, commit message, and where applicable the PR URL. Two pointer flags — **current** and **GitHub head** — are drawn on the entries they apply to. A

reviewer looking at the timeline can read off, without leaving the page, the sequence: when the prompt was first imported, what edits have been proposed, which were merged, when the last GitHub-side change landed, and where the two pointers currently rest.

Compare

The Compare action takes any two versions from the timeline and renders a side-by-side diff. The default selection is the current version against the GitHub head, because that pair answers the most common question — *what's different between what mowa is running and what's on GitHub*. The picker permits any two versions, including a currently-open suggestion against a previously-shipped version, which is the second most common question.

The diff is text-level. Edits inside the prompt body are visible at line granularity; structural intent is not surfaced. The reviewer is the one interpreting the diff against their knowledge of the system's downstream behavior.

What This Substrate Does Not Yet Do

The history timeline answers *what changed in our control surface and when*. It does not answer *why the production behavior differs*. Answering the latter in an automated way requires four capabilities that are not yet implemented:

- **A stored input set** that captures the kinds of requests the production system actually receives, with enough breadth to be representative.
- **Per-version replay** that re-runs the stored input set against an arbitrary historical prompt version under a controlled model and a controlled context, producing a behavioral record for that version.
- **Controlled substitution** that varies one of (prompt version, model version, retrieval context) while holding the others fixed, so the behavioral delta can be attributed to the dimension that was varied.
- **A baseline behavioral fingerprint** per version that gives the comparison something to compare against. This is the same fingerprint described in the companion note on the prompt provenance registry.

The design intent is that each of these is additive to the substrate that exists. Test inputs are already a first-class concept in the editor's Test tab; the missing piece is the discipline of curating a representative production-shaped set and storing it as a regression input. Per-version replay is mechanically straightforward once that input set is in place — the version content is already addressable. Controlled substitution and baseline fingerprints depend on the broader behavioral telemetry layer.

Position

We describe this surface as a substrate for diagnosis rather than a diagnostic engine. It is the persistence and presentation work that makes the diagnostic question askable. The engine that would automate the answer — counterfactual replay against version-pinned prompt and model and context — is design intent at the time of writing. Naming the gap explicitly is preferable to claiming a capability the product does not yet have.

Status

The history timeline and Compare view described in this note are implemented and in production use. Stored test-input sets are implemented per prompt and can be replayed against the live version, but per-version historical replay, controlled substitution, and automated counterfactual attribution are design intent, not shipped functionality.